# Exploring the Implementation of Several Key Slurm Inter-Cluster Features

Stephen Trofinoff
*CSCS – Swiss National Supercomputing Centre*
*Lugano, Switzerland*
*Email: trofinoff@cscs.ch*

***Abstract*—This paper describes the background and technical details associated with the exploration of and the implementation of a couple of new Slurm "inter-cluster" features. These features would extend some currently available functionality, such as job-chaining, to cross multiple independent Slurm clusters or in some cases, such as simultaneous job launch, add functionality that hitherto has not existed. The use of these features has been requested from our own users as well as other Slurm-based HPC labs; however, due to the array of questions and considerations posed by their various potential implementations have not yet been realized.**

***Keywords*-Slurm; inter-cluster feature; job-chaining; simultaneous job launch; slurmdbd; slurmctld; slurmctld-to-slurmctld communication**

## I. Introduction

Over the course of several years, both at our site (CSCS) and at others of which we were told, various instances have arisen where there was a need for some inter-cluster Slurm [1] features. These features would simplify or in some cases enable use cases for our various computing facilities and potentially make administering them easier. One prominent such request, was for the ability to chain a job to one or more jobs on a remote Slurm cluster. These features, of course, do not currently exist or are limited in their scope. For instance, a job can be submitted to a remote Slurm cluster but can not be "chained" to a job on another cluster since one Slurm cluster's controller has no knowledge of the jobs of another. Therefore, after various discussions, it was decided to start a small project at our site to explore the potential implementation of some of these features. The project is a work-in-progress with the aim of creating a working prototype. This paper and the corresponding presentation will discuss some of the work done thus far. This includes specifying the particular features chosen for examination, some issues related to their implementation and the current state of the work.

## II. Evolution of the Project

The first hurdle in this project, as with any project, was to define the scope; that is what specific features were to be targeted. With such a term as broad as "inter-cluster features", many features and work items could potentially be included, all valid and useful. However, as this is just an initial exploratory project and like most sites we need to balance our personal work loads, we tried to select a couple of features that were of particular interest to us and, time-permitting, one or two additional ones that other groups have expressed interest in. Thus for us, job chaining across clusters was our primary interest followed by the ability to simultaneously start jobs on different clusters (this was a request from a partner site).

The second hurdle was to try to organize the work and come up with ideas on how to actually implement it. Thus began a brainstorming phase where we (CSCS), SchedMD and LLNL shared some ideas on what exactly needed to be done. As can be imagined, there were multiple ideas; however, we selected one possible approach and began attempting to implement it to see how feasible it was.

The underlying idea, was that in order to implement some of the higher features mentioned above (e.g. job chaining) the individual Slurm clusters need to be aware of what specific system (Slurm cluster) a given job id is on. As the `slurmctld` is the "brain" of Slurm, containing all the information regarding jobs currently in the queue and is the entity in the Slurm system that physically produces the job id's, currently it only knows of its own job id's–not those of any other Slurm cluster. Therefore, the first technical issue to implementing these features was how to make clusters aware of on which cluster a job, with a given job id, was on. Currently, the job id is simply an unsigned 32-bit integer. One idea was to introduce a second integer that could represent the cluster. Although this has the advantage of leaving the entire original integer (and hence range of job id's) open for the cluster's jobs before wrapping occurs, it has the serious implementation drawback of necessitating the rewriting of all code throughout Slurm that deals with job_id's to be able to process a second integer. This could be perhaps mitigated somewhat by the use of a single integer but of twice the size (a 64-bit integer) but this would still require tracking down code throughout Slurm and potentially modifying how it handled this value.

In the end, it was decided to retain the single, current-sized, integer. Instead, it would have some other entity keep track of where a job was queued (on which system). This, again, lead to discussions on what methods to try. At one point, the creation of a new separate daemon that would be responsible for the creation of the Slurm job id's and doling them out to the various clusters was considered. It would also need to handle queries from the participating

clusters to determine where the job was. However, as some very early work was pointing out, this role was in some ways very similar to that of the existing slurmdbd. That is, it is an independent Slurm daemon that handles requests from various Slurm clusters at various key points during the job life-cycle (the Slurmdbd also handles requests from the the administrator and users via the sacct and sacctmgr commands). Therefore, it seemed natural to use it since it already had been designed to handle high volumes of communication from multiple clusters. It is also an optional Slurm entity and thus any job id feature based upon it should also be designed to be optional (something configurable, for instance, in the slurmdbd.conf file).

Once this decision was made there was the decision of which distribution scheme to use for the job id's. That is, should a cluster make a request to the slurmdbd EACH time it creates a job? Should each cluster be allocated a subrange of job id's, etc.? At first, we tried the latter. Again, there were different methods of implementation and choices along the way that could be made, each with their inherent benefits and drawbacks. As we were going to set aside a block of consecutive job id's for a given cluster and, at first, it was assumed that there could be an arbitrary number of clusters participating in this scheme we needed a methodology to divy the ranges. Since the feature for specifying min/max job id's already existed in Slurm, one idea tried early on was to have each cluster inform the slurmdbd, upon registration, what job id range it would like to use. On the one hand this would simplify some aspects of the design because it would mean that the slurmdbd would not need to have some sort of algorithm to determine the appropriate range size of job ids for a given cluster. However, it would mean that the slurmdbd would still need to receive the REQUESTED range from each cluster and then verify that there were no overlaps. Then, during its normal response to the cluster, it would inform it if it had a valid range or not. This, obviously, places the burden upon the system administrators or whoever configures Slurm on each cluster to ensure that they use proper ranges. It requires new information to be piggybacked on the registration request (job id range values) and required validation information. This code was actually implemented and did work but after more discussion it seemed that there were other and better methods of handling this issue. It also should be noted that with this method, there was a choice of whether to simply have a cluster blocked from using any inter-cluster feature if its range had an overlap or to add further logic to the `slurmdbd` to then correct an overlapped range, possibly truncating it or granting entirely different start and end points (all of which would, of course, be returned during the response).

The method preferred at the time by SchedMD was to have the controllers request, for each job, the job id to use. This has the advantage of the fact that there is already communication between the cluster's controller and the `slurmdbd` for each job and so this information could be potentially piggybacked. This method does, however, require a cluster to query the slurmdbd for job id's for which it needs information but is not aware of itself, just as in the other method. As this could lead to job ids being on random clusters, it would necessitate that a cluster almost always need to query the slurmdbd to find information about a job.

In the end, it was decided to go with a sort of hybrid approach where we would distinguish between a "normal" job with a locally generated job id and that of a special inter-cluster job with a job id generated by the `slurmdbd`. This would be done by use of a special reserve range of job id's for inter-cluster jobs such as all low job id's up to a certain threshold or, viceversa, all jobs above a certain job id threshold. For any job not within the reserved range, the local Slurm cluster controller would handle it as it normally would. That is, it would generate it in the same fashion and use all the same internal logic as it always had. However, for jobs within the special reserved range, they would receive their job id ultimately from the `slurmdbd`. In essence, this creates a second type of job as these jobs would be "visible" to other clusters because a cluster would know, inherently, that a job id was an inter-cluster job by virtue of the id itself. If it didn't have a job record for it, then it knows that it must obtain the location of the job from the `slurmdbd` which would then pass back the contact details of the controller for the cluster on which the job is running. It would still necessitate that the clusters contact the `slurmdbd` for queries regarding some jobs, but only inter-cluster jobs and only those not on its own cluster.

One of the implementation details that actually was common to most of these schemes including the latest was that the `slurmdbd` would maintain a new data structure. This structure would contain all the necessary controller contact information for each cluster of the grid. This currently consists of a hostname (in the form of an IP address string and a port number). With each of these it would associate the cluster name as a key index for table lookup. Additionally, in the earlier rendition where the clusters each requested specific ranges, the Slurmdbd would maintain range information (i.e. start/end values). Obviously, the latest approach was more efficient in this point, as well, as it no longer needed these fields.

One side question regarded how to handle clusters being added and dropped; that is, a dynamic set of participating clusters. This is a problem common to all of the afore-scribed methods. However, after discussions with SchedMD, it became clear that what was being envisioned was a more "static" set of clusters–a "grid" of clusters. Therefore, for the time being and the purpose of this paper, it will be assumed that we have a static grid with minimal changes in member states (registered/non-registered). It should be noted that although this issue does not affect the current design nearly as much as it had the design with distinct

job id ranges or designs with arbitrary clusters joining and leaving, there could still be some issues such as what to do if a cluster requests the location of a job only to learn that it is on a cluster that is currently down.

At this point, we had a basic scheme of enabling job id's, at least a select set of them, to be visible to multiple clusters and, more importantly, to be unique across all of them. This then laid the foundation for the implementation of the next phase of the work; that is, the chaining of jobs across clusters. Obviously, this involved numerous coding changes and again more decisions and choices but, with the groundwork laid out with the job id's above, it became much more straight-forward. Esssentially, the code that initially created the dependency field in the job record during its creation and the code to check if it were still unsatisfied had to be modified (`update_job_dependency`, `test_job_dependency`). These functions would handle normal job id's as they traditionally would but for inter-cluster job ids, if the record didn't exist on the local cluster it would contact the `slurmdbd` for its location and then communicate with the appropriate foreign controller to find out the status of the job. It should be noted that for an inter-cluster job id where the local `slurmctld` has a `job_record` for it, it would be handled as if it were a regular job.

## III. Current State

Currently, we have a basic implementation of the common job id range scheme and of inter-cluster job chaining. The code has undergone some basic function testing but would need to be further developed and refined in some areas to ensure that it would be robust enough for production use.

Although a number of changes, both large and small, had to be made to many different files, the API for the functionality, thus far implemented, was kept as simple as possible.

The `slurmdbd`'s configuration file has the added option `interClusterJobIdStart`. This is an unsigned 32-bit integer that denotes the start of the inter-cluster job id range.

The `slurmctld`'s configuration file now contains `GridClusters` and `ClusterIDMode`. The former is a list of Slurm cluster names participating in the grid. The latter simply indicates whether or not the cluster will participate in the grid; thus, providing the option to the administrator of whether to use the cluster in the grid or by itself as it traditionally would.

From the user perspective, thus far, the use model is perhaps even simpler still. In it, the user will identify, from the start by using the `--sicp` clause, whether a job should be a potential target of a remote job. This newly added clause to both the `sbatch` and `salloc` commands indicates to the Slurm system that it should receive a job id from the special reserved range. Other than this, the job is a normal job. However, now the user can, on another system (or the current one although that would be nothing new), specify a dependence upon this job id and have the remote job correctly target this original inter-cluster job. Note that the dependent job may also be an inter-cluster job for further job chaining across clusters. There is no change to the dependency syntax. The user still specifies a clause such as `--dependency=afterok:[job id]`. The only difference is that the job id would be from the reserved range and, thus, typically is much larger than other job ids.

In terms of which dependencies would be enabled for inter-cluster use, `after`, `afterany`, `afterok` and `afternotok` may be used. The `singleton` and `expand` dependency types were not implemented as it may not be needed nor make much sense (e.g. trying to "expand" a job's resources across clusters).

## IV. Implementation Details

The creation of the common job id range involved primarily modifications to the `slurmctld` and `slurmdbd` code with some additional minor modifications to some commands such as `sbatch` and `salloc` which needed to have the new option `--sicp` added. Conceptually, the logic flow is that a `slurmctld`, having been configured as a participating member of a Slurm grid, sends a message to the `slurmdbd` with its contact information. The `slurmdbd` then adds this contact information to a table stored in its internal memory and then returns this table along with an integer denoting the starting point of the common job id range (all job ids greater than or equal to this are in the reserved range) to the requesting cluster and all other participating clusters that are currently up. Thus, each `slurmctld` needs to be able to receive this update not only after it sends its own request but at arbitrary times afterwards as the clusters will not come online all at once. With this done, the `slurmdbd` and the respective controllers maintain a consistent table of Slurm cluster contact information and a consistent starting point of the common job id range.

Next, when a job on a specific cluster is being created with the option to run as an inter-cluster job, its `slurmctld` contacts the `slurmdbd` for the next job id in the common range instead of taking the next job id in its own normal local range. The `_set_job_id` function was thus modified to send the request to the `slurmdbd` for an inter-cluster job id when the user requests the job to be using `--sicp`. The `slurmdbd` upon receiving this message then picks the next inter-cluster job id and creates an entry in another table in its internal memory. This table correlates the job id with a given cluster name for future lookup. The `slurmdbd` then sends a message back to the requesting `slurmctld` with the new job id.

When another job is being created, for instance on a second Slurm cluster (ClusterB), which has a dependence upon the previously created inter-cluster job queued on the

first Slurm cluster (ClusterA), the `slurmctld` of ClusterB sends a message to the `slurmdbd` to find out on which cluster the remote target is running. (Note: It does this only if it does not have a record for the job itself, in which case the inter-cluster job would be local and there would be no need for any special processing). The `slurmdbd` looks the job id up in its table of used inter-cluster job ids and retrieves the cluster name on which it was queued. Then it determines at which index this cluster is at in the grid table and sends a response message with this single integer, the index, back to the requesting `slurmctld`. At this point, the `slurmctld` of ClusterB uses this index to retrieve the contact information from its own local copy of the grid table and stores this information directly in the dependency structure of the job record corresponding to the dependent job. In this way, each time the scheduling cycle comes to testing this job's dependencies, it will have the contact information readily available so that it can directly contact the appropriate foreign controller. At such time, the `slurmctld` of ClusterB would contact the `slurmctld` of ClusterA requesting the status of the target job. ClusterA returns the status which is a simple integer. ClusterB then places this value in a dummy job record so that it could pass through all of the same logic as it normally would (the same internal test macros). Thus, from this point through the end of the scheduling cycle, the traditional logic is used. For instance, if an `afterok` dependency type was used and the status is a successful completion, the dependency would be removed and the job would be eligible to run (barring any other dependencies that still may exist). Likewise, if the target failed or was cancelled then the dependent job would be cancelled as its dependency was unfullfilled. It should be noted that this style of notification of the status of the foreign job is a "pull"-style and not a "push"-style request. This means that when a target job completes, it does not notify all possible controllers of this fact but simply is handled as it typically would with its state being updated. As the foreign dependent jobs from the various remote clusters come to the point of checking their dependencies, then they contact the target's controller for the information. Thus, in this scheme, there is often a delay between when a target completes and when the corresponding changes in foreign dependencies result. This is typically under a minute in testing due to the frequency of the scheduling cycle.

(See Figure 1 for a schematic representation of the implementation.)

## V. Open Questions and Other To-do's (A Partial List)

One open question is what to do if the foreign job record is already expunged from its controller when a remote dependent job tries to test its dependency by contacting the foreign controller for the status of the target job. If the job record has already been expunged, then this currently is treated as if it had failed (just for the purposes of the remotely dependent job). Thus, if there had been an `afterok` job dependency, it would be cancelled thinking that the target had failed. Likewise, an `afternotok` dependent job, would be incorrectly launched. Should a counter be added to a `slurmctld`'s job record of how many foreign jobs may be dependent upon it? Should we add such a counter to the `slurmdbd`'s data structure instead? For instance, in both of these cases, there could be a field in the corresponding record, initialized to zero and then incremented once for each job that claims a dependence upon it. As these jobs complete, the counter could be then decremented. Once the counter reaches zero, the record could then be cleared out of memory. Using the counter approach with the records being stored in the `slurmdbd` would seem to be the preferred of these two variants as all the clusters already communicate job completion status to the `slurmdbd`. Furthermore, it would allow the `slurmctld` on each cluster to follow its existing logic when it comes to handling its `job_records` (which are already quite large and complex).

A still better approach, perhaps, would be simply to have the foreign controller, once contacted for the status of one of its inter-cluster jobs that has already been expunged, contact the `slurmdbd` itself for the status. At that point, the `slurmdbd` could either check its own table of inter-cluster jobs for a status value (if the status were to be added) or simply query the database and pass back the result. This would have the advantage of already communicating with the database in which such information is stored indefinitely (the basic purpose of the database). The downside would be if there were many such requests due to there existing many dependent jobs that didn't request the status until after the expunging of the controller's record, this could then cause significant overhead.

In the end, how often this would occur would depend upon various conditions such as the configuration of the various clusters (how long it retains job records after completion),

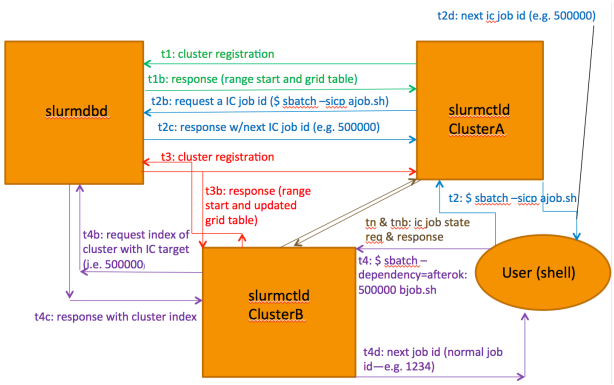


Figure 1. Idle vs Busy timings for the `SUMMARY` and `INVENTORY` methods on the 8-node TDS.

the size of the workloads across the various systems and use patterns (e.g. how many jobs are being chained) amongst other things.

As stated, the `slurmdbd` needs to keep track of inter-cluster job id's in use. However, this raises the question of how long should it retain this information? This is, in some ways, related to the question of what to do about job records for remote targets that have already been expunged as, if we use this record as a potential back up, it would need to persist for longer. Then there would exist the same issues, such as, should it simply be removed after a set amount of time after job completion, or should it possibly have a counter? As job id wrapping wasn't explicitly handled yet with this new type of job id (Slurm does handle it for the traditional scheme) perhaps records could be removed once wrapping is detected although this would seem undesirable.

With regard to the dependency records of remote target jobs, if there is more than one job for the given type of dependency and it is not on the same remote target cluster, then this would potentially fail as it currently wasn't taken into account. Perhaps the solution here would be to have the dependency record store a list of contact information such that the order of the entries would correspond to the order of the remote target job ids associated with that dependency type.

Another current limitation is that the `slurmdbd` must be started before all participating clusters. The order of the start up of the clusters is not important but if the `slurmdbd` is started after any of them, those clusters that began before it will not be seen as part of the grid.

Similarily, nothing is specifically done in the event that a cluster goes down nor is the contact information for the backup controllers currently stored.

The `slurmdbd`'s grid table has a maximum size that is currently hard-coded and unchangeable. This needs to be changed so that it can expand the table as the number of cluster registrations increases beyond the current size.

In addition to the various open questions stated above there are still other parts of the original project remaining to be implemented. This includes the simultaneous job launch feature as well as some updates to several commands such as `squeue` and `scontrol` to show inter-cluster job information.

On a final note, the current work is based upon the now outdated Slurm version 2.5.4 as the project had initially started before version 14.03.x was released. Thus, the work should be ported to a newer version of the Slurm code for greater relevance.

## VI. Conclusion

Due to multiple requests for job-chaining across clusters and other new inter-cluster features currently not available in Slurm, a project was started to explore some possible implementations. Although still a work-in-progress, a basic prototype has been produced that successfully demonstrates that it is possible to do at least job-chaining between clusters. In doing this, some key design points have already been addressed such as how to implement a mechanism for allowing one Slurm cluster to recognize the job id's of another Slurm cluster.

Having reached this point, the project aims to further enhance and refine the current modifications and to continue to explore the additional functionality, such as simultaneous job launch, cited in the original proposal.

## VII. Glossary

- cluster: A Slurm cluster.
- foreign controller: A `slurmctld` (control daemon) for another Slurm cluster external to the one in question.
- grid: A fixed and predefined set of independent Slurm clusters that will be able to use the various new inter-cluster features described in this paper.
- inter-cluster feature: Features in Slurm functionality that are designed to work across different independent Slurm clusters.
- SICP: Acronym for CSCS's project to explore the implementation of various Slurm inter-cluster features. (Slurm Inter-Cluster Project) Currently used as the identifier for the new `sbatch` and `salloc` option to specify that a job should be designated an inter-cluster job.

## Acknowledgment

Thanks to Moe Jette, Danny Auble and the rest of the team at SchedMD as well as Don Lipari of LLNL for their collaboration.

## References

[1] [Online]. Available: http://www.schedmd.com/